Secure static content delivery for Content Distribution Network using Blockchain technology

Pier Paolo Tricomi
Student ID: 1179740

***Abstract*—A Content Distribution Network (CDN) is a new kind of network with the goal to distribute services and content spatially relative to end-users, providing high availability and high performance. In order to reach this goal, several replicas of the Origin Server are used, but trust issues are now present both between Servers and among Clients and Servers. In this work a new method to provide secure static content delivery is presented, which makes use of a Blockchain, a growing technology with the capability to ensure reliability and trust without a central authority.**
Moreover, a prototype of the system has been developed on Ethereum private network, in order to test its feasibility. The test shows the goodness of the system, and the ability to create a new content distribution model over the Internet.

1. Introduction

Content distribution is one of the most important aspects of the Internet. To improve the distribution the Content Distribution Networks (CDNs) are born. CDNs provide high availability and high performance because many replicas (Edge Servers) of the Origin Server are spatially distributed to faster fulfill a request. Many services like CoralCDN [1] and CoDeeN [2] provide the ability to create a CDN starting from an Origin Server, replicating content and distributing it to end users from the best Edge Server, usually the geographically nearest.

In this scenario, two new trust issues are introduced. First of all, an attacker could modify the content from the Origin Server to the Edge Server. If the indirectly attacked Edge Server serves that modified content, it will be labeled as a misbehaving replica. Secondly, if an Edge Server is not directly managed by the owner of the Origin Server, it can serve different content, such as stale content or outright modified content, adding ads for example. Furthermore in this architecture the Origin Server is a single point of failure. If the Origin Server is compromised all the Servers misbehave.
The main contribute of this work is two-fold.
First, a new architecture to overcome the trust issue between Servers is suggested, maintaining the CDN properties. Second, it is shown how the system provides a secure method to deliver static content, ensuring the integrity with small effort on Clients and Network.

2. Related Works

Checking the integrity of data retrieved from untrusted Servers is a crucial problem. The typical approach for preventing content tampering between Clients and Servers is to encrypt the end-to-end connection, for example using the SSL protocol. This negates the functionality of the CDN, because it is always required a connection to the Origin Server, thus the network is no more distributed. For static content, Merkle tree authentication [3] is a possible solution, where the Server signs the content which is verified by the Client. Another possible solution are digital rights management schemes, which allow Clients to verify data from an untrusted Server [4]. Both these solutions are useless if the Server is compromised. For dynamic content, [5] [6] propose the use of XML-based rules for managing the content, but it has to be limited to be easily verified by a Client.
In peer-to-peer CDNs these problems are even more significant, because Clients can serve content too. LOCKSS [7] uses a voting system for content integrity. Repeating the execution to detect misbehavior has been used in Rx [8] and in Vigilante [9] as a method to discover respectively bugs and worms. These approaches imply significant overhead on the Client side. In Pioneer [10] the verify effort is on the dispatcher, a trusted platform, but since the proof of correctness is extremely time sensitive it is not suitable for large scale systems. Finally the Repeate and Compare system [11], which is for both static and dynamic content, requires the repetition of the content to another replica and compares the results to detect misbehaving replicas. However, the process has significant overhead, and the verification requests may stress the network, thus only a fraction of the content is verified.
In this work Blockchain technology is used. In [12] a video sharing system using a Blockchain is proposed, but implementation details are not provided. The structure of a Blockchain ensures trust between untrusted nodes without a central authority [13]. A system of content distribution built on top of a Blockchain inherits all its benefits, making sharing and verifying content simple.

3. Blockchain Technology

After the rise of Bitcoin [13], its underlying structure, the Blockchain technology, has been applied to a variety of

usecases ranging from authentication [14] (using Ethereum and Smart Contracts) to medical reports [15]. Blockchain is based on an append-only ledger, a growing list of records (called blocks) which are linked and secured using cryptography. The ledger can be viewed by all participating nodes and the updates are permitted only after the consensum of the network. Each block typically contains a cryptographic hash of the previous block, a timestamp and transaction data. The presented system relies on Ethereum, an open-source, public, Blockchain-based distributed computing platform.

3.1. Ethereum and Smart Contracts

Ethereum [16] is described as "a transactional singleton machine with shared-state". The distributed ledger can be viewed as a distributed virtual machine which records transactions and state changes. The most important feature of Ethereum is encoding Smart Contracts.
Smart Contracts describe functions, which can be called by nodes participating in the Blockchain, and states which can be changed by the nodes. The functions run in the Ethereum Virtual Machine (EVM), a Turing complete computer able to execute code of any complexity. The contract can use memory and storage to save data: memory is volatile while storage persists over time. The last can be changed, but all the changes will be recorded onto the ledger.
In the presented sytem, thanks to its persistence and (im)mutability, storage will be used to store and share content.

3.2. Benefits

In this paragraph, benefits of using Blockchains are discussed. Since the presented content delivery system is built on top of a Blockchain, it will inherit all its benefits.

- **Decentralization:** a consensus mechanism is used to agree on the validity of transactions, thus there is no need for a trusted third party;
- **Transparency and trust:** Blockchains are shared and visible to all the partecipants. This makes the system transparent and as a result trust is established;
- **Immutability:** once the data has been written into the Blockchain, it is extremely difficult to change it back. In Ethereum if the state is changed all the updates are stored in the ledger. If only a few users are allowed to change the state, the state will remain immutable as long as they don't change it;
- **High availability:** since the system is based on several nodes in a peer-to-peer network, and the data is replicated and updated on each node, the network as a whole continues to work even if nodes leave or become unavailable;
- **Highly secure:** all the transactions on a Blockchain are cryptographically secured and provide integrity.

4. System Design

In this section the developed system to distribute and check integrity of the content is presented. A part of the content is called object, and each object of the content can be verified. Firstly an high-level overview is shown. Secondly more details about the architecture are provided. Lastly benefits and drowbacks of the system are discussed.

4.1. Overview

Figure 1 shows an high-level view of how the presented system distributes the content on the Servers and permits to Client to check integrity.

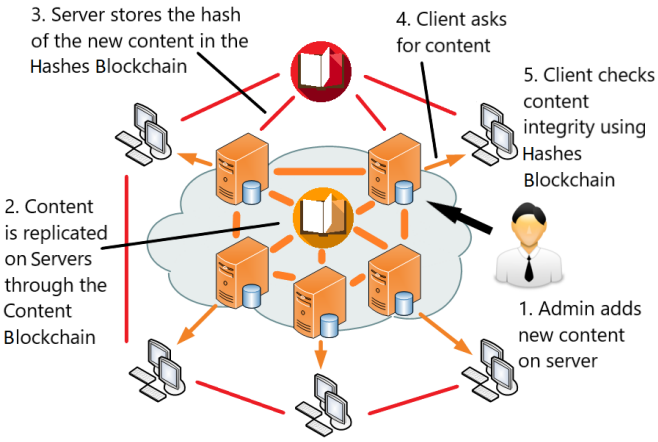

Figure 1. Overview of the content delivery and check system.

The system wants to resolve trust issues between Origin and Edge Servers, since the latter can receive altered content, and between Servers and Clients, as Servers can send content which differs from the requested one. The key idea is to use two different Blockchains: the *Content Blockchain* and the *Hashes Blockchain*.

4.1.1. Content Blockchain. The proposal is to build the CDN on a Blockchain, the *Content Blockchain*. This means that each Edge Server is a node of the Blockchain wherein content will be stored. The Origin Server is no more required, since it is possible to add content from every node (Edge Server) as long as it is authorized, and the content will automatically be distributed on each Server thanks to the Blockchain technology. To authorize the upload of an object of the content an user system is used. Since the Blockchain is always visible to every participant, and every participant has a public address, which consists in a public and private key pair, storing the addresses of who can upload the content in the Blockchain is enough, and after having checked the identity of who is requesting the upload, the transaction can be executed. Two kind of users are permitted: Admin and Superadmin. Admins can upload/remove/edit the content while Superadmins can also create other Admins.

4.1.2. Hashes Blockchain. The *Hashes Blockchain* is the key in checking the integrity of received object of the content. Thanks to hash functions checking message integrity is really simple [17]. An hash function is any function that can be used to map data of arbitrary size to data of fixed size. A cryptographic hash function allows one to easily verify that some input data maps to a given hash value, but if the input data is unknown, it is extremely difficult to reconstruct it by knowing the stored hash value. These property are very useful for the goal. Every time an object of the content is uploaded, its hash is calculated by the Server and stored in the *Hashes Blockchain*, where the partecipants are both Servers and Clients. After the Client receives the requested object it can easily check its integrity calculating its hash and comparing it with the one stored in the Blockchain. If the hashes are equal the object is right, otherwise something wrong happend. Since storing the hash of the object requires the consensum of the network, the hash is always correct, and there is no possibility for a Server to change it without noticing the others nodes.

4.2. Typical Usage

A typical use case is shown in Figure 1. First an Admin wants to upload a new content. After the system checks and authorizes him, a new upload request is sent among the *Content Blockchain*. When the network approves the request, a copy of the object is distributed on each Server (according to Blockchain technology) and its hash is calculated and stored in the *Hashes Blockchain*. A Client can ask for that object at any time. When the object is received the Client calculates its hash with the same hash function used by the Server and compares the obtained hash with the calculated one. If they are equal the integrity is verified, otherwise the Client can reject the content and ask to another Server.

4.3. Benefits and Drawbacks

In many distributed systems multiple entities maintain their own databases and data sharing can become very difficult due to the disparate nature of the systems. Since a Blockchain can serve as a single shared ledger among interested parties, this can reduce the complexity of managing the whole system, removing the possibile processes of verification, reconciliation, and clearance. The content on all the Servers is always correct and updated, the problem of a replica receiving wrong content from the main Server is avoided.
Many of cited systems like Pioneer [10] and Repeat and Compare [11] require sending multiple request to the Server (to get the content and to check it) with the possibility to overcrowd the network, whereas others system causes significant overhead on the Client. In the presented system the traffic needed to check integrity is correlated to the maintaining of the Blockchain itself, which should not be really relevant, and the effort on the Client is minimum as shown in Section 5. When Client wants to verify the obtained data it just has to calculate an hash, and check the result. Since the checking process is unexpensive for Clients and not stressful for the network, each object of the content can be verified, unlike the other systems where the detection is probabilistic. Furthermore, even the misbehaving Servers detection is simpler thanks to the Blockchain technology.
Another point of strenght is the removal of the Origin Server; an Admin can simply upload content from his home and it will be distributed among the Servers (peers). Now there is not a single point of failure, since if a Server is compromised the network will know it. The attacker has to steal the private key of an Admin to take control of a Server, which is difficult, and it is a problem present in any other solution.
In this system we have some drawbacks aswell. First of all, all the Servers must have all the content that the system can serve, which can be expensive in some scenarios. However, if the *Content Blockchain* only contained the more requested content or the URIs of the resources, it would be possible to reduces the costs. Secondly, the Clients must have a proxy or some similar mechanism to automatically check the integrity, but this is not a big deal since it can be done with a simple browser add-on for example. Lastly, keeping updated the Blockchain has some costs for the network, but precise tests should be executed to evaluate if this is a real problem, and in a real enviroment the uploading process could be slow due to the mining operation.

5. Implementation

The system relies on Ethereum Blockchain and Smart Contracts. Each Smart Contract can have multiple instances. The contracts are written in Solidity, a high level language designed to encode contracts in Ethereum. The system was tested on a private Ethereum testnet, which is useful for beginners because the use of money (gas) is not involved. To create the network *geth* was used, the command line interface for running a full ethereum node implemented in Go. The network was composed by few nodes, one Admin and one Superadmin, and a single miner. The difficulty of mining a new block was low, so that testing could be faster. Different Smart Contracts are developed for *Content Blockchain* and *Hashes Blockchain*. The deploy of contracts was made by Remix IDE, which allows to connect to the private testnet via JSON-RPC. In the next sections the main functionalities of the system are explained.

5.1. Administration System

Only Admins are allowed to manage the objects of the Content. The Administration system is implemented using one istance of the `AdminAddresses` Smart Contract. In its storage there is a Map in which all the Admins and Superadmins are present. It maps the address of a user with its privileges (Admin or Superadmin). If an address is not present in the Map then it is not able to manage the Content. The contract thus allows:

- to check if a user is an Admin (if its address is in the Map);

- to get the user privileges (Admin or Superadmin);
- to set Superadmin privileges to a user (only if the requester is a Superadmin);
- to add or delete an Admin (only if the requester is a Superadmin).

5.2. Content Managing

The *Content Blockchain* is where the Content on the Servers is stored. The `StoredContent` Smart Contract allows to create and edit an object. This means that each object of the Content is an instance of `StoredContent`. Each object has the following attributes (stored in the Storage):

- the owner, that is the uploader Admin;
- the name;
- the description;
- the timestamp of the upload;
- the data;
- the hash of the data.

5.2.1. Creation. To add an object the constructor of the contract requires its name, the data and the description. After having checked if the uploader is an Admin (using the `AdminAddresses` Smart Contract), it calculates the hash of the data using *keccak256* function, which is an alias of *sha3* supported by Ethereum. The hash is 32 bytes long. When the instance of the contract is mined the object is distributed on all nodes of the *Content Blockchain*, according to Blockchain technology. Since all the nodes of the Blockchain has the same ledger, the Content present in the storage is equal in each node. This resolves the trust issue of sharing Content between Origin and Edge Servers.
To find the object in the Blockchain a Map of all objects is used, but a block explorer to find the requested object could be implemented to save space. When the object is created, its addres must be added to the Map. The `ContentMap` is the Smart Contract which contains the Map of the objects. In the storage of its unique instance there is the map of all the objects, wherein the key is the name of the object. Mapped to the key there are the address of the object, its hash, and the names of the other objects related to it. Merkle Tree are used to save space on Clients to check integrity of the objects, thus each object is related to other three objects in our scenario, where the Merkle Tree is composed by four leafs (further information are provided in Section 5.4). The contract allows:

- to insert an object in the Map;
- to check if an object is present in the Map;
- to edit the name of an object. This implies that the `StoredContent` instance representing that object is called to change its name, and that the name is changed in the related names of its related objects;
- to get the address of the object;
- to associate a new object to a specific name;
- to remove the object from the Map;

Note that an object cannot be deleted from the Blockchain, since that technology doesn't permit it, but its entry in the Map can be removed. Each function of the contract firstly check if the requester is an Admin using the `AdminAddresses` Smart Contract.
Finally the information for integrity checking has to be stored in the *Hashes Blockchain*. The Server uses the `ObjectsMerkleTree` Smart Contract (which is better explained in 5.4) to create or update the Merkle Tree in which the hash of the object has to be stored.

5.2.2. Edit. The `StoredContent` Smart Contract permits to edit the attributes of an object except the owner, the timestamp and the hash, since they are always calculated by the Smart Contract at each update. Every change of attributes implies the update of the Map of the `ContentMap` contract, and the changes are allowed only after having checked if the requester is an Admin, using the `AdminAddresses` Smart Contract. If the data is changed, the Server uses the `ObjectsMerkleTree` Smart Contract to create or update the Merkle Tree in which the hash of the object has to be stored.

5.3. Request of an object

Client can request an object of the Content in an ordinary way. The Server will call the `ContentMap` Smart Contract to get the address of the object, then it uses the `getData` method of the `StoredContent` instance located at the fetched address to get the data of the object, and it finally sends the object to the client in an ordinary way.

5.4. Integrity checking

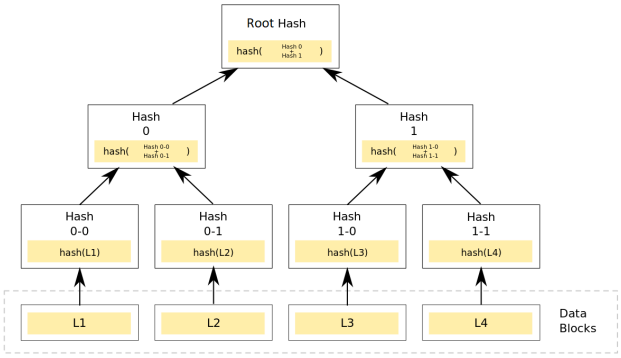

Figure 2. Example of Merkle Tree. In the root is stored the combination of the hashes stored in the leafs.

To check the Content integrity the *Hashes Blockchain*, formed by MerkleTrees in order to save space, is used. As shown in Figure 2, a Merkle Tree is a tree in which every leaf node is labelled with the hash of a data block and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. Instead of storing one hash per

object, using a Merkle Tree of four leafs, for example, is possible to store one hash every four objects. The number of objects per tree can be increased, with a lot of space saving. In Table 1 the required space on Clients using a Map with all the objects versus a Merkle Trees of four objects is shown.

TABLE 1. EXAMPLE OF REQUIRED SPACE ON CLIENTS.

Number of objects	Simple Map	Merkle Tree
100'000	3,05 Mb	0,76 Mb
1'000'000	30,52 Mb	7,63 Mb
30'000'000	915,52 Mb	228,88 Mb

MerkleTrees are implemented using the `ObjectsMerkleTree` Smart Contract. Each instance of this contract has in its storage the root hash of the Merkle Tree composed by a group of four related objects. The contract also provides methods to calculate, store and check the root hash.
To check if an object is intact, when the Clients requests that object, the Server sends it with also the hashes of the related objects. Then it is sufficient to calculate the root hash of the new Tree composed by the calculated hash of the received object and the hashes of the related objects in the right order, and lastly comparing the result with the root hash stored in the *Hashes Blockchain*. It is almost impossible for the Server to change the object and sending fake hashes for matching the hash stored in the Blockchain, since the hash functions are robust to this attack.
To understand which Merkle Tree an object belongs the `MerkleTreeMap` contract has been used, which maps names to `ObjectsMerkleTree` instance addresses, but in the future a block explorer should be used.

6. Future Works

The presented system has been developed on Ethereum, but it limits the code execution of Smart Contracts imposing gas payment. Some instruction like loops can be very expensive, thus it may not be the best enviroment for the scope. A new ad-hoc Blockchain could be implemented for the content distribution, to better fit the requirments. Another improvment could be replacing the data of the content in the *Content Blockchain* with an URI or another type of storage to permit Servers to contain also other objects, depending on the situation. The idea of using a secure URI would lead to the creation a global Blockchain for content distribution, which different services and websites can entrust with their content. For example, every service can have its own Blockchain, and in the global Blockchain could be links to these Blockchains. In the end, the presented system is only for static content, a better implementation could be thought for sharing dynamic content too.

7. Conclusion

In the presented system a solution is found for trust issues in a Content Distribution Network. Thanks to the growing Blockchain technology and its security characteristics, it is possible to securely distribute content among Servers, and a Client can check if the requested content is intact. Many problems of exiting systems like strong efforts on the Network or Clients are solved, and the prototype built on the private Ethereum testnet shows the feasibilty of such system. The use of Merkle Tree leads to space saving, and with other smaller improvements the bootstrap of the system should be a real possibility.

References

[1] M. J. Freedman, E. Freudenthal, and D. Mazieres, "Democratizing content publication with coral." in *NSDI*, vol. 4, 2004, pp. 18–18.

[2] L. Wang, V. Pai, and L. Peterson, "The effectiveness of request redirection on cdn robustness," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 345–360, 2002.

[3] R. J. Bayardo and J. Sorensen, "Merkle tree authentication of http responses," in *Special interest tracks and posters of the 14th international conference on World Wide Web*. ACM, 2005, pp. 1182–1183.

[4] A. Adelsbach, M. Rohe, and A.-R. Sadeghi, "Towards multilateral secure digital rights distribution infrastructures," in *Proceedings of the 5th ACM workshop on Digital rights management*. ACM, 2005, pp. 45–54.

[5] C.-H. Chi and Y. Wu, "An xml-based data integrity service model for web intermediaries," in *Proc. 7th IWCW*, 2002.

[6] H. K. Orman, "Data integrity for mildly active content," in *Active Middleware Services, 2001. Third Annual International Workshop on*. IEEE, 2001, pp. 73–77.

[7] P. Maniatis, D. S. Rosenthal, M. Roussopoulos, M. Baker, T. J. Giuli, and Y. Muliadi, "Preserving peer replicas by rate-limited sampled voting," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 44–59.

[8] F. Qin, J. Tucek, J. Sundaresan, and Y. Zhou, "Rx: treating bugs as allergies—a safe method to survive software failures," in *Acm sigops operating systems review*, vol. 39, no. 5. ACM, 2005, pp. 235–248.

[9] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, "Vigilante: End-to-end containment of internet worms," in *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5. ACM, 2005, pp. 133–147.

[10] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, "Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems," in *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5. ACM, 2005, pp. 1–16.

[11] N. Michalakis, R. Soulé, and R. Grimm, "Ensuring content integrity for untrusted peer-to-peer content distribution networks," in *Proceedings of the 4th USENIX conference on Networked systems design & implementation*. USENIX Association, 2007, pp. 11–11.

[12] J. Kishigami, S. Fujimura, H. Watanabe, A. Nakadaira, and A. Akutsu, "The blockchain-based digital content distribution system," in *Big Data and Cloud Computing (BDCloud), 2015 IEEE Fifth International Conference on*. IEEE, 2015, pp. 187–190.

[13] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[14] R. Sundararajan and S. K. Shukla, "Online identity and authentication using blockchain."

[15] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "Medrec: Using blockchain for medical data access and permission management," in *Open and Big Data (OBD), International Conference on*. IEEE, 2016, pp. 25–30.

[16] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[17] G. Tsudik, "Message authentication with one-way hash functions," in *INFOCOM'92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE*. IEEE, 1992, pp. 2055–2059.